Copyright Notice

The following manuscript

EWD 215: A Case against the GO TO Statement

was published as a letter entitled

Go-to statement considered harmful

in *Commun. ACM 11* (1968), 3: 147–148. It is reproduced here by permission.

A Case against the GO TO Statement.

by Edsger W.Dijkstra
Technological University
Eindhoven, The Netherlands

Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to effectuate the desired effect; it is this process that in its dynamic behaviour has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visiulize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost best to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded

as the descriptions of single actions) it is sufficient to point in the program text to a point between two successive action descriptions. (In the absence of go to statements I can permit myself the syntactic ambiguity in the last three words of the previous sentence: if we parse them as "successive (action descriptions)" we mean successive in text space, if we parse as "(successive action) descriptions" we mean successive in time.) Let us call such a pointer to a suitable place in the text a "textual index".

When we include conditional clauses ("if B then A"), alternative clauses ("if B then A1 else A2"), choice clauses as introduced by A.R.Hoare ("case[i] of(A1, A2,.....,An)") or condional expressions as introduced by J.McCarthy ("B1 → E1, B2 → E2,....., Bn → En"), the fact remains that the progress of the process remains characterized by a single textual index.

As soon as we include in our language procedures we must admit that a single textual index is no longer sufficient: in the case that a textual index points to the interior of a procedure body the dynamic progress is only characterized when we also give to which call of the procedure we refer. With the inclusion of procedures we can characterize the progress of the process via a sequence of textual indices, the length of this sequence being equal to the dynamic depth of procedure calling.

Let us now consider repetition clauses (like "while B repeat A" or "repeat A until B"). Logically speaking, such clauses are now superfluous, because we can express repetition with the aid of recursive procedures. For reasons of realism I don't wish to exclude them: on the one hand repetition clauses can be implemented quite comfortable with present day finite equipment, on the other hand the reasoning pattern known as "induction" makes us well equipped to retain our intellectual grasp on the processes generated by repetition clauses. With the inclusion of the repetition clauses textual indices are no longer sufficient to describe the dynamic progress of the process. With each intry into a repetition clauses, however, we can associate a so-called "dynamic index", inexorably couting the ordinal number of the corresponding current repetition. As repetition clauses (just as procedure calls) may be applied nestedly, we find that now the progress of the process can always be uniquely characterized by a (mixed) sequence of textual and/or dynamic indices.

The main point is that the value of these indices are outside programmer's control: they are generated (either by the write up of his program or by the dynamic evolution of the process) whether he wishes or not. They provide independen coordinates in which ti describe the progress of the process.

Why do we need such independent coordinates? The reason is -and this seems to be inherent to sequential processes- that we can interpret the value of a variable only with respect to the progress of the process. If we wish to count the number, "n" say, of people in an initially empty room, we can achieve this by increasing "n" by 1 whenever we see someone entering the room: in the in-between moment that we have observed someone entering the room but have not yet performed the subsequent increase of "n", its value equals the number of people in the room minus one!

The unbridled use of the go to statement has as an immediate consequence that it becomes terribly hard to find a meaningful set of coordinates in which to describe the process progress. Usually, people take into account as well the values of some well chosen variables, but this is out of the question becuase it is relative to the progress that the meaning of these values is to be understood! With the go to statement one can, of course, still describe the progress uniquely by a counter counting the number of actions performed since program start (viz. a kind of normalized clock). The difficulty is that such a coordinate, although unique, is utterly unhelpful: in such a coordinate system it becomes an extremely complicated affair to define all those points of progress where, say, "n" equals the number of persons in the room minus one!

The go to statement as it stands is just too primitive, it is too much an invitation to make a mess of one's program. One can regard and appreciate the clauses considered as bridling its use. I do not claim that the clauses mentioned are exhaustive in the sense that they will satisfy all needs; but whatever clauses are suggested (e.g. abortion clauses) they should satisfy the requirement that a programmer independent coordinate system can be maintained to describe the process in a helpful and manageable way.

It is hard to end this article with a fair acknowledgement: am I to judge by whom my thinking has been influenced? It is fairly obvious that I am not uninfluenced by Peter Landin and Christopher Strachey and that I do not regret their influence upon me. Finally I should like to record (as I remember it quite distinctly) how Heinz Zemanek at the pre-ALGOL meeting in early 1959 in Copenhagen quite explicitly expressed his doubts whether the go to statement should be treated on equal syntactic footing with the assignment statement. To a modest extent I blame myself for not having then drawn the consequences of his remark.